Self-Managed Leasing for Distributed Systems

Kevin Bowers
Renssalaer Polytechnic Institute
bowerk@rpi.edu

Kevin Mills, Steve Quirolgico, and Scott Rose
National Institute of Standards and Technology
{kmills, steveq, scottr}@nist.gov

ABSTRACT
We describe an adaptive algorithm that enables a distributed Jini enabled system, given a fixed allocation of resources, to vary lease periods to achieve the best responsiveness.

Keywords
algorithm, self-managing, performance, Jini, leasing

1. INTRODUCTION
Distributed systems require strategies to detect and recover from failures. One commonly used strategy employs a leasing mechanism, where a node grants a leaseholder access to a resource for a limited time (the lease period). Once the resource is no longer needed, the leaseholder may relinquish its lease. If the resource is needed beyond the original lease period, then the leaseholder can renew the lease by requesting additional lease periods. If the leaseholder does not renew before expiration of the lease period, the lease grantor assumes leaseholder failure and terminates the lease.

Choosing an appropriate lease period entails tradeoffs among resource utilization, responsiveness, and number of leaseholders. We explore these issues in the context of service-discovery protocols, which allow distributed software components to discover each other and compose themselves into assemblies. Though several service-discovery protocols currently exist [e.g., 1-3], [5] we selected Jini Network Technology [1] to demonstrate our ideas, because leasing plays a central role in registering Jini services.

2. JINI LEASING
Jini defines an architecture that enables clients and services to rendezvous through a third party, known as a lookup service. A Jini service registers a description of itself with each discovered lookup service.

A registering component requests registration for duration L_R, which may be accepted at time T_G for a granted lease period $L_G \leq L_R$. L_R or may be *any*, which allows any value for L_G. To extend registration beyond L_G, registering components must renew the lease prior to an expiration time $T_E = T_G + L_G$; otherwise, registration is revoked. This cycle continues until a Jini component cancels or fails to renew a lease. Lookup services assign L_G within a configured range, $L_{MIN} \leq L_G \leq L_{MAX}$. While a granted lease may not be revoked prior to T_E, lookup services may deny any lease request.

We can analyze performance of a Jini leasing system. Let S_R be lease-request size, S_G be lease-grant size, and N be number of leaseholders. Typically, a leaseholder and lookup service exchange one request-grant pair per renewal cycle, with rate $1/L_G$ Hz. Assuming identical L_G assigned for each lease, bandwidth use (B) can be estimated as: $B=(N/L_G)\cdot(S_R+S_G)$. Assuming constant S_R and S_G, B increases linearly with N and decreases exponentially with L_G. Another metric, responsiveness, R, measures the latency with which lookup services can detect leaseholder failure. Assuming uniformly distributed failure times, then expected responsiveness is $R = L_G/2$; thus, R is independent of N, but B and R are related through L_G.

These relationships can be used to constrain and predict behavior of a leasing system. For example, assume known requirements for R and B. The responsiveness equation can be rewritten to determine L_G [i.e., $L_G = 2R$]. Then, using L_G, the bandwidth equation can be transformed to find maximum system size [i.e., $N_{MAX} = (B \cdot L_G)/(S_R + S_G)$]. With this information, lookup services could grant lease periods $\leq L_G$ to ensure required responsiveness, deny requested leases that would consume an excess share of bandwidth, and deny requests for leases once N reaches N_{MAX}.

3. A SELF-ADAPTIVE ALGORITHM FOR JINI LEASING
Assuming a leasing system must consume at most bandwidth B and guarantee minimum average responsiveness R_{BEST}, a lookup service can grant a maximum lease period $L_{MAX} = 2R_{BEST}$. Given B, S_R, and S_G, we can determine a maximum lease-renewal rate $G = B / (S_R + S_G)$. For minimum system size, $N_{MIN} = 1$, the lookup service can grant a minimum lease period $L_{MIN} = 1/G$. While this value for L_{MIN} respects the bandwidth constraint, other factors should be considered. For example, at $L_{MIN} = 1/G$ leaseholder processing burden

might prove unacceptable. Instead, a leasing system might constrain maximum responsiveness (R_{WORST}), giving a minimum lease period $L_{MIN} = 2R_{WORST}$. Knowing N, a lookup service may select a suitable granted lease period from a range ($L_{MIN} \leq L_G \leq L_{MAX}$) using a simple algorithm. First, compute $L_G = N/G$. If $L_G > L_{MAX}$, then deny the lease; otherwise, if $L_G < L_{MIN}$, then set $L_G = L_{MIN}$. Assigning L_G with this algorithm permits a leasing system to constrain B and guarantee minimum average responsiveness (R_{BEST}), while providing the best responsiveness achievable (up to R_{WORST}) as N varies over $1..N_{MAX}$.

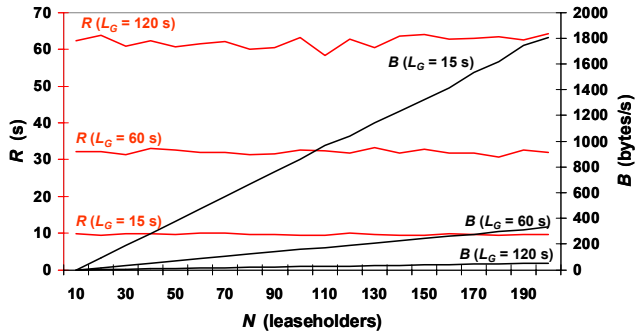

Fig. 1. System responsiveness (*R*) – left-hand y-axis – and bandwidth usage (*B*) – right-hand y-axis – for three granted lease periods (*LG* = 15 s, 60 s, and 120 s) as system size increases (*N* = 10 to 200 leaseholders).

4. SIMULATION RESULTS

We coded an SLX discrete-event simulation [5] model of Jini to confirm our analysis and to investigate dynamic behavior of our self-adaptive algorithm. We conducted simulation experiments, varying N from 10..200 and L_G from 15..300 s in 15-s increments. We used $S_R = 128$ and $S_G = 32$ bytes. Figure 2 shows simulated results for average B and R when L_G = 15 s, 60 s, and 120 s. The simulation confirms our analyses: (1) B increases linearly with N for a given L_G and decreases exponentially with L_G for a given N and (2) $R = L_G/2$, independent of N. Next, we simulated our adaptive leasing algorithm. Figure 3 illustrates how the algorithm constrains B while improving R as system size decreases.

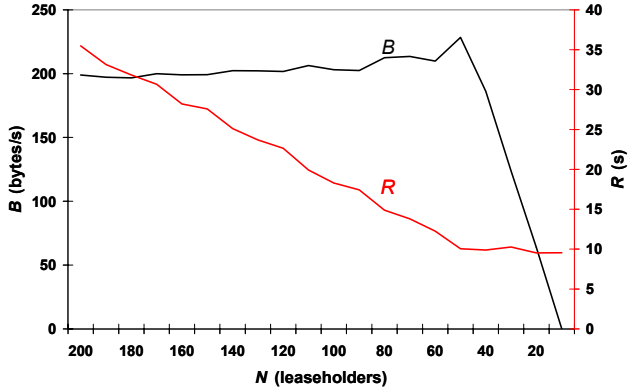

Fig. 2. Responsiveness (*R*) – left-hand y-axis – and bandwidth usage (*B*) – right-hand y-axis –as system size decreases (*N* = 200 to 0 leaseholders).

5. IMPLEMENTATION

These promising results led us to implement our adaptive algorithm in "reggie", a publicly available implementation of a Jini lookup service. Administrative actions occur through the *RegistrarAdmin* interface, which is not part of the Jini core specification, but a Sun extension to Jini. The *RegistrarAdmin* interface allows basic monitoring, configuration, and control of an operational lookup server just as if it were any other type of Jini-enabled service.

To implement self-managed leasing, we modified the "reggie" server code to assign lease-grant times (L_G) based on an administrator-assigned policy specified by two target values: worst-case average failure responsiveness (R_{WORST}) and average bandwidth (B) allocated to lease renewal transactions. We added a collection of access methods to the *RegistrarAdmin* interface, allowing a Jini client to view: the current L_{MIN}, L_{MAX}, and L_G, the number of leaseholders (N) on the server, the instantaneous average bandwidth (B_{AVG}) consumed by lease renewals, and the instantaneous average failure responsiveness (R_{AVG}).

Results from our live experiment are similar to the results we obtained from simulations. For example, the behavior of B_{AVG} (Bandwidth) and R_{AVG} (Responsiveness) were similar to the simulation results when services were added, then removed from the network.

6. FUTURE WORK

Given the performance tradeoffs and implementation costs, we conclude that our simple adaptive leasing algorithm can yield useful performance properties at little cost. We argue that our adaptive leasing algorithm should also apply to similar systems that employ leasing for resources, such as UPnP event subscriptions [2]. The modifications are done on the resource provider side, and any system that allows for flexible lease times should be able to take advantage of this algorithm.

7. REFERENCES

[1] Jim Waldo. "The Jini™ architecture for network-centric computing", *Communications of the ACM*, July 1999.

[2] Universal Plug and Play Device Architecture, Version 1.0, 08 Jun 2000 10:41 AM. © 1999-2000 Microsoft Corporation. All rights reserved.

[3] Ken Arnold et al, The Jini Specification, V1.0 Addison-Wesley, 1999. The latest version is available on the web from Sun.

[4] James O. Henriksen, "An Introduction to SLX™," *Proceedings of the 1997 Winter Simulation Conference*, ACM, Atlanta, Georgia, December 7-10, 1997, pp. 559-566.

[5] A. Ninan, P. Kulkarni, P. Shenoy, K. Ramamiritham, and R. Tewari. "Performance: Cooperative Leases: Scalable Consistency Maintenance in Content Distribution Networks", *Proceedings of the eleventh International Conference on World Wide Web*, May 2002.